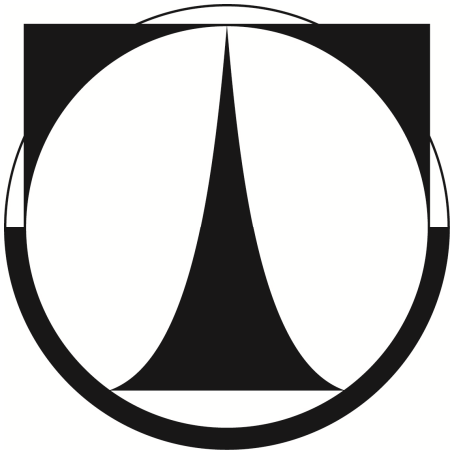
TECHNICKÁ UNIVERZITA V LIBERCI
FAKULTA MECHATRONIKY, INFORMATIKY A MEZIOBOROVÝCH STUDIÍ



BAKALÁŘSKÁ PRÁCE

květen 2012 Tomáš Červinka

TECHNICKÁ UNIVERZITA V LIBERCI
FAKULTA MECHATRONIKY, INFORMATIKY A MEZIOBOROVÝCH STUDIÍ

Studijní program: B2646 - Informační technologie

Studijní obor: 1802R007 - Informační technologie

SIMULÁTOR PROCESORU PIC

SIMULATION OF THE PIC PROCESSOR

Autor: Tomáš Červinka

Vedoucí práce: Ing. Tomáš Martinec, Ph.D.

Pracoviště: Ústav mechatroniky a technické informatiky

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2011/2012**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Tomáš Červinka**
Osobní číslo:	**M10000136**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Simulace procesoru PIC**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Seznamte se s existujícím simulátorem hybridních obvodů a s architekturou procesorů PIC.

2. Navrhněte model vybraného procesoru PIC a způsob jeho implementace v simulátoru.

3. Realizujte navržený model a otestujte ho na vhodných příkladech.

Rozsah grafických prací:	**dle potřeby dokumentace**
Rozsah pracovní zprávy:	**cca 30–40 stran**
Forma zpracování bakalářské práce:	**tištěná/elektronická**

Seznam odborné literatury:

[1] **Dokumentace k simulovaným obvodům.**

[2] **Dokumentace k použitým vývojovým prostředím.**

Vedoucí bakalářské práce:	**Ing. Tomáš Martinec, Ph.D.** Ústav mechatroniky a technické informatiky
Konzultant bakalářské práce:	**Ing. Přemysl Svoboda** Ústav mechatroniky a technické informatiky
Datum zadání bakalářské práce:	**14. října 2011**
Termín odevzdání bakalářské práce:	**18. května 2012**

prof. Ing. Václav Kopecký, CSc.
děkan

Technická univerzita v Liberci - 14 -

doc. Ing. Petr Tůma, CSc.
vedoucí ústavu

V Liberci dne 14. října 2011

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Poděkování

Především bych chtěl poděkovat panu Ing. Tomáši Martincovi, Ph.D., který byl mým vedoucím již u bakalářského projektu a nyní je vedoucím mé bakalářské práce. Pokud jsem si nevěděl rady v určité fázi práce, vždy jsem se na něj mohl obrátit a prakticky obratem mi poradil a to i formou osobních konzultací.

Dále bych chtěl poděkovat své rodině za podporu při studiích na Technické univerzitě v Liberci.

Anotace

Tématem této bakalářské práce je simulace mikrokontroléru PIC16F877A v prostředí simulátoru HyCiSim, který bude sloužit studentům jako pomůcka k předmětu PHS (počítačový hardware a software). Při návrhu součástky je potřeba jistá znalost simulátoru HyCiSim a návrhového prostředí pro elektronické obvody KiCad. Úvodní kapitoly práce se proto zabývají popisem knihoven a metod simulátoru HyCiSim a poskytují návod k vytvoření grafické podoby součástky v prostředí KiCad. Hlavní část tvoří podrobný popis mikrokontroléru PIC16F877A a převod jeho vlastností do podoby programového kódu za účelem simulace.

Annotation

The topic of this bachelor thesis is simulation of microcontroller PIC16F877A in environment of the HyCiSim simulator, which is intended as a tool for teaching PHS subject (computer hardware and software). To design an electric component, knowledge of HyCiSim simulator and design environment for electronic circuit KiCad is needed. Therefore, the introductory chapter deals with description of HyCiSim library and method of HyCiSim simulator and offers instructions on how to design a graphic representation of component in KiCad environment. The main part consists of detailed description of microcontroller PIC16F877A and transfering in programming code for simulation purpose.

1 Obsah

Seznam obrázků

Seznam tabulek

2 Úvod

Mikroprocesory řady PIC firmy Microchip Technology jsou díky své nízké ceně, široké škále uplatnění a kvalitním návrhářským nástrojům hojně využívány jak v průmyslu, tak mezi zájmovými návrháři. Mikroprocesory PIC jsou harvardské architektury, která má oddělenou programovou a datovou paměť. To umožnuje používat jinou šířku sběrnice pro přenos dat a jinou pro instrukce. V rámci architektury RISC (zkratka z anglického „Reduced Instruction Set Computer“) mají mikroprocesory PIC redukovanou instrukční sadu, která je zaměřena na jednoduchost a vysokou optimalizaci strojových instrukcí. Díky tomu je většina instrukcí vykonána v jednom instrukčním cyklu, který trvá čtyři takty mikroprocesoru. Tato práce se konkrétně zabývá mikroprocesorem PIC16F877A a jeho softwarovou simulací v prostředí simulátoru HyCiSim.

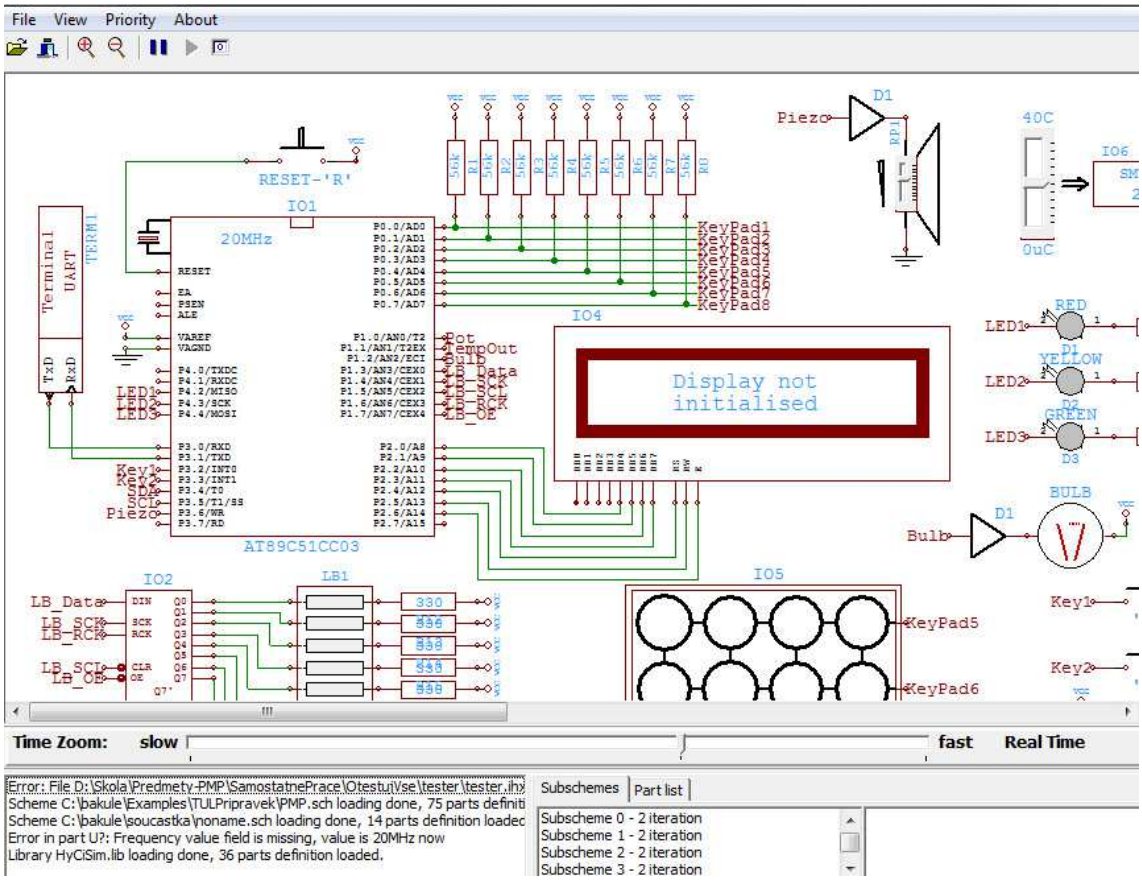
První část práce seznamuje s již zmíněným simulátor elektronických obvodů HyCiSim a v jednoduchosti popisuje vlastní knihovny simulátoru, které je nezbytné znát pro návrh nové součástky. Na ukázce kódu elementárních součástek rezistoru a LED diody jsou použity základní metody těchto knihoven pro správnou funkci součástky v elektronickém schématu.

Hlavní kapitolou je detailní návrh součástky mikroprocesoru PIC16F877A. Ta se zabývá elektronickými vlastnostmi jednotlivých vstupně/výstupních portů, softwarovým návrhem paměťové struktury, metodami přístupu k datům a jednotlivými instrukcemi z instrukční sady. Krom toho je PIC16F877A vybaven periferními obvody jako jsou čítače/časovače, které mohou být zdrojem přerušení. Obsluhou těchto přerušení se zabývá závěrečná část práce.

3 Simulátor HyCiSim

Simulátor HyCiSim slouží k simulaci elektronických obvodů. Naprogramoval ho Ing. Tomáš Martinec, Ph.D. v prostředí Delphi jazyka Pascal. Celou simulaci je možno sledovat v reálném čase s ohledem na výkon zařízení, na kterém je spuštěna. Případné posunutí oproti hodnotě reálného času je zobrazeno v procentech při běhu simulace. Pro návrh součástky slouží čtyři základní knihovny. Knihovna GraphicsElement definuje základní tvary pro vykreslení součástky. Pomocí BasicObjects jsou naprogramovány elektrické vlastnosti součástky a její další specifické funkce. Scheme načítá součástky z knihovny součástek a rozparsovává schéma na podschémata, což je důležité pro výpočet ustáleného stavu elektrického obvodu. Jednotlivé události vyvolané schématem řídí knihovna Scheduler.

Zmíněnou knihovnu součástek lze vytvořit v prostředí pro návrh elektronických obvodů KiCad. Jednotlivé součástky se v KiCadu vytvoří v nástroji Eeschema, kde je možno definovat polaritu pinů, jméno a textové pole hodnot součástky. Výsledek je vyexportován v textové podobě pro možnost načtení součástky v externích programech.

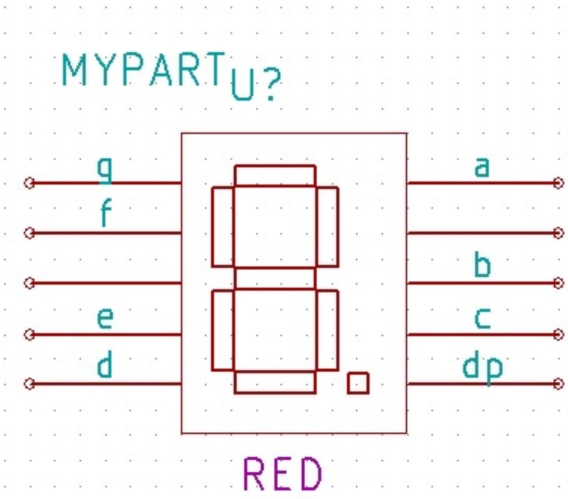


Obr. 1 - Prostředí simulátoru HyCiSim

3.1 Výstup z návrhového prostředí KiCad

Následující kód představuje výstup grafické definice součástky sedmi-segmentového displeje.

```
# MYPART
DEF MYPART U 0 0 N Y 1 F N
F0 "U" -108 672 60 H V L TNN
F1 "MYPART" -108 672 60 H V R CNN
F4 "2V0" 670 333 60 H I C CNN "Voltage"
F5 "RED" 2 -132 60 H V C CNN "Color"
F6 "50" 692 122 60 H I C CNN "Persistance"
DRAW
S -200 550 250 -50 0 1 4 N
S -135 75 -90 235 0 1 0 N
S -135 280 -90 440 0 1 0 N
S -90 30 70 75 0 1 0 N
S 70 75 115 235 0 1 0 N
X f 2 -500 350 300 R 50 50 1 1 I
X b 8 550 250 300 L 50 50 1 1 I
X ~ 9 550 350 300 L 50 50 1 1 I
X a 10 550 450 300 L 50 50 1 1 I
ENDDRAW
ENDDEF
#End Library
```



Obr. 2 - Příklad součástky vytvořené v KiCad

3.1.1 Popis formátu souboru součástky

DEF name reference unused text_offset draw_pinnumber draw_pinname unit_count units_locked option_flag

name = jméno součástky v knihovně

reference = reference (U, R, IC .., následně U3, U8, R1, R45, IC4...)

unused = 0 (rezervováno)

text_offset = posun textu se jménem pinu

draw_pinnumber = Y (zobrazí se čísla pinu) or N (čísla pinů jsou skryta)

draw_pinname = Y (zobrazí se jména pinu) or N (jména pinů jsou skryta)

unit_count = počet částí (nebo sekcí) v balíčku součástky, maximálně 26 (části jsou pojmenovány A-Z)

units_locked = = L (části nejsou identické a tudíž nemohou být zaměněny), F (části jsou identické, a proto mohou být zaměněny), toto se využívá je-li unit_count > 1

option_flag = volitelný příznak N (normální součástka), P (součástka typu zdroj)

F0 reference posx posy text_size text_orient visibile htext_justify vtext_justify

F1 name posx posy text_size text_orient visibility htext_justify vtext_justify

reference = reference (U, R, IC .., následně U3, U8, R1, R45, IC4...)

name = jméno součástky v knihovně

posx, posy = pozice popisky součástky

text_size = velikost zobrazeného textu

text_orient = orientace textu (V = vertikální, H = horizontální (implicitně))

visible = zobrazení popisku (I = neviditelný, V = viditelný (implicitně))

htext_justify = horizontální zarovnání textu (L = vlevo, R = vpravo, C = na střed)

vtext_justify = vertikální zarovnání textu (T = k vrchu, B = do spod, C = na střed)

ALIAS je definován, pokud má součástka přidělené jméno aliasu

Format: ALIAS jméno1 jméno2 jméno3…

DRAW

Obsahuje seznam grafických elementů součástky a jejich pinů, kde každý řádek definuje jeden element. Řádek začíná písmenem značícím typ elementu. Základními parametry elementu jsou:

posx, posy = pozice elementu

unit = číslo části, pokud jich má součástka více

convert = v případě, že má součástka více variant grafického schéma, 0 indikuje žádné varianty

thickness = tloušťka čáry

fill = vyplnit barvou (F = vyplněn barvou popředí, f = vyplněn barvou pozadí, N = nevyplněn (implicitně))

Seznam grafických elementů

A - oblouk

A posx posy radius start_angle end_angle unit convert thickness fill startx starty endx endy

posx, posy = střed kruhu (skládajícího se z oblouků)

radius = poloměr kruhu

start_angle = počáteční úhel oblouku v desetinách stupně

end_angle = konečný úhel v desetinách stupně

startx, starty = koordinace počátku oblouku

endx, endy = koordinace konce oblouku

C - kruh

C posx posy radius unit convert thickness fill

posx, posy = střed kruhu

radius = poloměr kruhu

P - multilinka

Multilinka se sestává z několika bodů.

P point_count unit

point_count = počet spřažených párů

S - obdélník

S startx starty endx endy unit convert thickness fill

startx, starty = počáteční roh obdélníku

endx, endy = koncový roh obdélníku

T - text

T direction posx posy text_size text_type unit convert text

direction = směr textu (0 = horizontální, 900 = vertikální (implicitně))

text_size = velikost textu

text = text k zobrazení

X - pin

X *name num posx posy length direction name_text_size num_text_size unit convert electrical_type pin_type*

name = jméno pinu

num = číslo pinu

posx = pozice na ose x

posy = pozice na ose y

length = délka pinu

direction = směr orientace pinu, R – vpravo, L - vlevo

name_text_size = velikost textu jména pinu

num_text_size = velikost textu čísla pinu

electrical_type = elektrický typ pinu (I = vstupní, O = výstupní, B = obousměrný, T = třístavový, P = pasivní, U = nespecifikovaný, W = vstup zdroje, w = výstup zdroje, C = otevřený kolektor, E = otevřený emitor, N = nepřipojen)

pin_type = grafický typ pinu (N = skrytý, I = invertovaný, C = hodinový vstup/výstup, IC = invertovaný hodinový vstup/výstup, L = pro nízké napětí, CL = Clock Low, V = výstup při programování při nízké napěťové úrovni, F = sestupná hrana, NX = nelogický)

3.2 Knihovny simulátoru HyCiSim

3.2.1 BasicObjects

Tato knihovna definuje jednotlivé součástky elektronického schématu, včetně spojovacích vodičů, uzlů obvodu a míst, kde se stýká více vodičů. Knihovna vytvoří jak elektrické vlastnosti součástky pro použití ve schématu, tak její grafickou podobu, kterou je dále možno upravovat. Tato knihovna dále obsahuje metody pro vyvolání událostí, které slouží pro reakci součástky na akce jako je: dvojklik, stlačení klávesy atd. Pro konkrétní součástku lze uložit a při příštím použití načíst poslední konfiguraci.

MaxPinCount – konstanta maximálního počtu pinů součástky (defaultně 100)

MaxJunctionsInSubscheme - maximální počet pinů připojených do jednoho bodu (defaultně 100)

h = 1e-1 (konstanta přesnosti) - určuje rychlost a stabilitu výpočtu ustáleného stavu

TPart = class(TBaseObject)

- *základní třída pro odvození všech součástek do simulace*

Fields : TObjectList

- *obsahuje popisky součástky, např.: hodnoty odporu, napětí atd.*

PinList : array[0..MaxPinCount-1] of TPartPin

- *seznam pinů součástky*

PinCount : Integer

- *počet pinů dané součástky*

OnChangeActive : Boolean

- *tato proměnná určuje, zda-li bude součástka během simulačního kroku reagovat na změnu napětí na jednom z vývodů (defaultně false)*

3.2.1.1 Funkce a metody:

procedure LoadConfig(FileName : String); virtual;

- *pomocí této funkce součástka načítá uložené hodnoty z konfigurace při spuštění schématu*

procedure SaveConfig(FileName : String); virval;

- *funkce pro uložení aktuálního stavu při zavírání schématu nebo celého programu*

procedure InitializePins(count : Integer);

- *tato metoda nastavuje v metodě Create počet pinů součástky*

procedure AddToInfoList(s : String);

- *touto metodou je možné napsat zprávu do simulačního programu*

procedure Draw(paper : TCanvas; Erase, Highlighted : Boolean); virtual;

- *nakreslí součástku na prázdnou plochu, pokud je Highlighted true pak ji zvýrazní, pokud je Erase true, pak ji smaže*

procedure ReDraw(paper : TCanvas); virtual;

- *tato funkce se vola v případě, že je nutné překreslit součástku, nejprve je nutné součástku smazat*

function IsInPartArea(X, Y : Integer) : Boolean; virtual;

- *funkce určující, jestli bude součástka reagovat na kliknutí myši*

procedure ConnectPin(n : Integer; Jun : TJunction);

- *pomocná metoda pro parsování schématu*

function RelationBetweenPins(Pin1, Pin2 : Integer) : Boolean; virtual;

- *rozdělení pinů součástky do podschématu, zda-li na sebe mají vliv při výpočtu ustáleného stavu*

function GetCurrent(pin : Integer) : TFloat; virtual;

- *funkce pro výpočet proudu na pinech součástky, počítá se podle ní ustálený stav*

procedure Reset; virtual;

- *tato metoda se volá ze schématu, pokud uživatel stiskne tlačítko pro reset simulace - musí se modifikovat, pokud si součástka má něco pamatovat (např. stav tlačítka apod.)*

Metody vyvolané událostmi:

procedure OnLeftClick(X, Y : Integer); virval;

procedure OnRightClick(X, Y : Integer); virtual;

procedure OnDoubleClick(X, Y : Integer); virtual;

procedure OnKeyPress(Key : Char); virtual;

procedure AfterChange(pin : Integer); virtual;

- *tato metoda se volá po přepočtu nového ustáleného stavu, podmínkou je nastavení proměnné OnChangeActive na hodnotu true*

3.2.2 GraphicsElement

Tato knihovna obsahuje grafické prvky a jejich umístění ve schématu. Každá součástka se skládá z několika takových prvků. Pokud součástka mění svou podobu na základě měnících se hodnot schématu nebo svých parametrů, je možné její grafické prvky překreslovat metodami knihovny BasicObjects (ReDraw) a například ji tak rozsvítit apod. Jednotlivé prvky pro vykreslení součástky lze vyčíst z knihovny součástek vytvořené v programu KiCad. Základní grafickou třídou knihovny GraphicsElements je TGraphicsElement. Z ní jsou odvozeny další třídy tvarů: TGECircle, TGEPolyline, TGERectangle, TGERectangle, TGEText, TGEPin.

3.2.3 Scheme

Knihovna Scheme definuje hlavní objekt TScheme, který načítá do paměti všechny knihovny součástek (umístěné v adresáři s programem). TScheme umí otevřít schéma a rozparsovat ho, tzn. vytvoří příslušné objekty a vazby mezi nimi. Další funkcí je zprostředkování komunikace programu s ostatními objekty schématu.

Metody:

procedure Clear;

- *smaže schéma*

procedure LoadLibrary(FileName : String);

- *načtení knihoven se součástkami*

procedure LoadScheme(FileName : String);

- *načtení schématu*

procedure LoadConfig(FileName : String);

- *načte konfiguraci součástek, volá se hned po načtení schématu*

procedure SaveConfig(FileName : String);

- *uloží konfiguraci součástek, volá se před zavřením schématu*

procedure OnLeftClick(X, Y : Integer);

procedure OnRightClick(X, Y : Integer);

procedure OnDoubleClick(X, Y : Integer);

procedure SetPaper(P : TCanvas; X, Y : Integer);

- *nastavení plochy pro vykreslování*

procedure Evaluate;

- *přepočet schématu, provede se jen na začátku, pak je řízeno součástkami*

Funkce pro parsování schématu:

function PointIsOnWire(X, Y : Integer; W : TWire) : boolean;

function PinIsInJunctions(S : TSubscheme; X, Y : Integer) : Boolean;

procedure AddWireToJunction(Jun : TJunction; W : TWire);

procedure AddJunctionToSubscheme(originalS : TSubscheme; Jun : TJunction; newS : TSubscheme);

procedure ParseScheme;

- *provede rozparsování schématu na podschémata a nastaví všechny vazby*

procedure CreateSubschemes(s : TSubscheme);

procedure Draw;

- *nakreslí schéma na plochu*

3.2.4 Scheduler

Knihovna Scheduler obsahuje typy událostí a třídu TScheduler, která obsahuje metody k obsluze těchto událostí. Lze přes ni řídit průběh simulace, včetně časové osy. Většina událostí se dostane k vykonání ihned, u některých lze nastavit zpoždění.

Seznam událostí

TEvent = class(TBaseObject)

- *základní třída událostí (prázdná), ostatní události jsou z ní odvozené*

  atributy: EventType : Integer, Time : Double

TChangePinEvent = class(TEvent)

- *událost změny napětí na pinu, volá metodu OnChangePin*

  atributy: PartPin : TPartPin

TRecountSchemeEvent = class(TEvent)

- *přepočet schématu (zavolá sadu událostí pro přepočet podschématu)*

  atributy: Scheme : TScheme

TRecountSubschemeEvent = class(TEvent)

- *vynutí přepočet subschématu*

  atributy: Subscheme : TSubScheme

TRedrawSchemeEvent = class(TEvent)

- *událost pro překreslení schématu*

TRedrawPartEvent = class(TEvent)

- *vynucené překreslení součástky*

TPartRequestEvent = class(TEvent)

- *volá OnMyRequest pro danou součástku, součástka může mít více požadavků (ID)*

  atributy: Part : TPart, ID : Integer

TPauseEvent = class(TEvent)

- *zastavení simulace, znovuspuštění tlačítkem Start*

TReconfigureSubschemesEvent = class(TEvent)

- *provede nové rozparsování schématu, používá se pokud dojde k velkým změnám schématu a nové rozparsování by mohlo zrychlit výpočty*

TScheduler = class

- *TScheduler je třída pro práci s událostmi*

  Scheme : TScheme;

  EventList : TObjectList;

- *seznam událostí k vyřízení*

  MaxEvents : Integer;

- *maximální počet událostí k vyřízení*

  StartTime, ActTime, SimulationTime : Double;

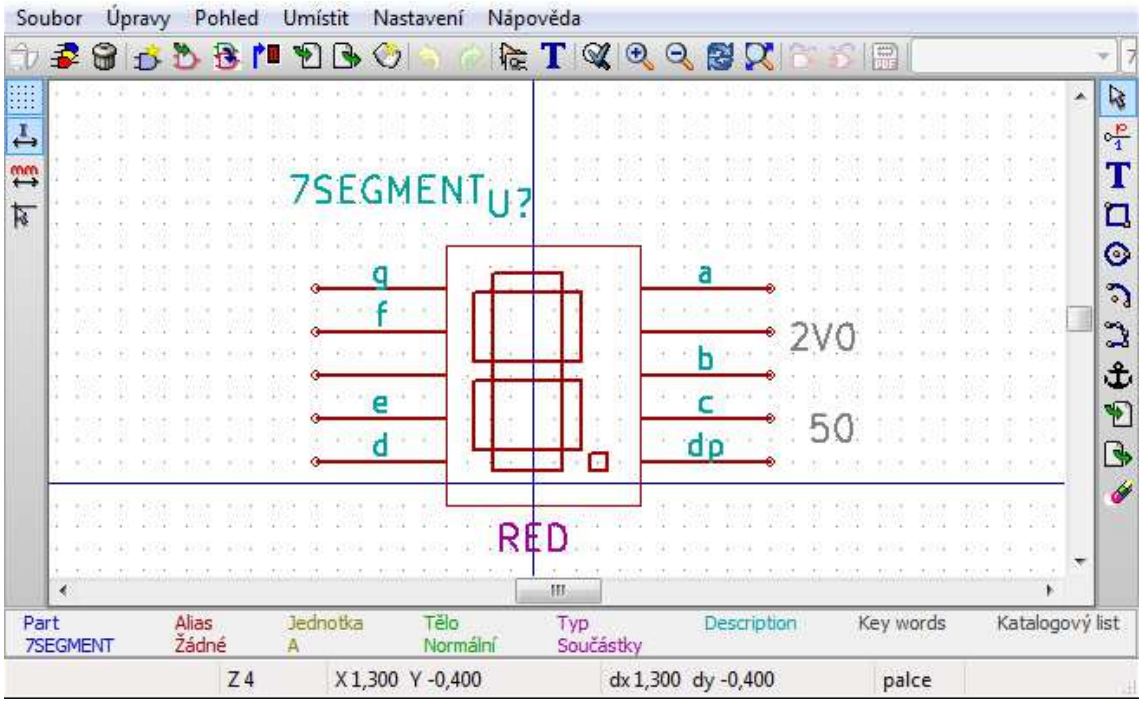
- *čas začátku simulace, aktuální čas (reálný), simulační čas*

4 Návrh součástky

4.1 Grafická podoba součástky

4.1.1 Návrhové prostředí KiCad

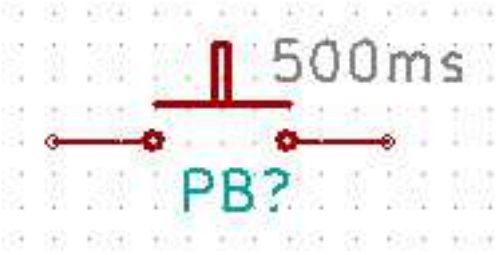
Prvním krokem při návrhu součástky je vytvoření její grafické prezentace. K tomu slouží již zmiňované prostředí pro návrh elektronických schémat a součástek KiCad. Součástku lze přidat otevřením editoru schémat Eeschema a následným spuštěním editoru knihoven. Souřadnicová síť slouží k narýsování vizuální podoby součástky pomocí tvarů na panelu nástrojů. Ten obsahuje i piny (pokud záleží na pořadí pinů, je nutné je očíslovat), které lze dále specifikovat (vstupní, výstupní, obousměrný pin) poklepáním na příslušný pin. Pro popis parametrů součástky slouží editační pole, které se vyvolá poklepáním na součástku nebo klikem na ikonu "T". Po uložení do nové nebo stávající knihovny je součástka hotová a připravená pro použití v simulátoru HyCiSim.



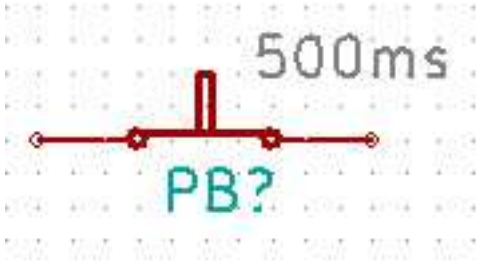
Obr. 3- Návrhové prostředí KiCad

4.1.1.1 Proměnný vzhled součástky

Některé součástky mají tu vlastnost, že mění svůj vzhled (např. tlačítko má dva dvě podoby: sepnuto a rozepnuto). První možnost jak toho docílit je vytvořit dvě vizuální podoby přímo v programu KiCad.



Obr. 5 - Rozepnuté tlačítko



Obr. 4 - Sepnuté tlačítko

Pojmenujeme-li při návrhu v KiCad jednu grafickou značku tlačítka v popisném poli součástky „PUSHBUTTON“ a druhou „PUSHBUTTON#ON“. Můžeme po kliknutí na tlačítko měnit jeho podobu ve schématu. Tlačítko tedy bude mít nadefinované dva stavy: OnSymbol a OffSymbol typu TPartSymbol což je třída, která načítá součástku z knihovny součástek. Nese tedy parametry jako je jméno (Name) apod.

```
type
  TPushButton = class (TPart)
    StateOn : Boolean;
    OnSymbol, OffSymbol : TPartSymbol;
    .
    .//zde jsou definovány metody tlačítka
    .
end;
```

```
procedure TPushButton.InitializeSymbol;
var
  i : Integer;
begin
  Inherited;

  OnSymbol := nil;
  for i := 0 to PartsLibrary.ItemsCount - 1 do
    if TPartSymbol(PartsLibrary.Items[i]).Name = SymbolName +
'#ON' then OnSymbol := TPartSymbol(PartsLibrary.Items[i]);

  if Symbol = nil then
    begin
      AddToInfoList('Error in part ' + Reference + ': Symbol for
On not found');
      OnSymbol := Symbol;
    end;

  OffSymbol := Symbol;
end;
```

V ukázce kódu inicializace tlačítka jsou pomocí inherited děděny vlastnosti obecné třídy součástky TPart, která obsahuje Symbol (instance třídy TPartSymbol). Úkolem této metody je načíst z knihovny obě podoby tlačítka. Procházením všech součástek knihovny (PartsLibrary.ItemsCount) se hledá podoba tlačítka sepnuto. Jméno součástky (Name) je tvořeno řetězcem z popisného pole součástky definovaného v programu KiCad až po znak „#", za kterým je možno definovat rozšiřující parametr, v tomto případě „ON". Pokud takový symbol není nalezen, přidělí se součástce implicitní podoba „PUSHBUTTON" uložená v Symbol.

Další možnost změny vzhledu součástky je její překreslení přímo ve schématu pomocí základních tvarů knihovny GraphicsElements.

```
procedure TLEDDiode.ReDraw(paper: TCanvas);
var
  p : Integer;
begin
  inherited;
  p := 70 div ZOOM;
  if Light then Paper.Brush.Color := LEDColor else
     Paper.Brush.Color := clSilver;
  Paper.Pen.Color := clBlack;
  Paper.Pen.Width := 1;
  Paper.Brush.Style := bsSolid;
  Paper.Ellipse(PosX div ZOOM - p, PosY div ZOOM - p, PosX div
ZOOM + p, PosY div ZOOM + p);
end;
```

- *V této ukázce je simulace rozsvícení nebo zhasnutí LED diody. Opět jsou zděděny vlastnosti obecné součástky TPart, máme tedy přístup k souřadnicím součástky ve schématu (PosX, PosY). Pokud má být LED dioda rozsvícena, překreslí se její značka stejně velkou elipsou, která je však vybarvená barvou svítící LED diody.*

Poslední možností jak měnit vzhled součástky je vyplnění vybraných prvků součástky barvou.

```
for i := 0 to Symbol.GraphicsElements.ItemsCount - 1 do
  begin
    if (Symbol.GraphicsElements.Items[i] is TGERectangle) then
      if (Symbol.GraphicsElements.Items[i] as
TGERectangle).Width = 0 then
        begin
         if j < 8 then Shape[j] :=
           (Symbol.GraphicsElements.Items[i] as TGERectangle);
           inc(j);
        end;
  end;
```

- *Tento kód prochází všechny grafické prvky součástky a hledá prvek obdélníku (TGERectangle) o zadané šířce čáry, kterou lze v programu KiCad nastavit. Tímto si lze vyselektovat například segmenty pro rozsvícení a uložit si je do pole pro lepší přístup.*

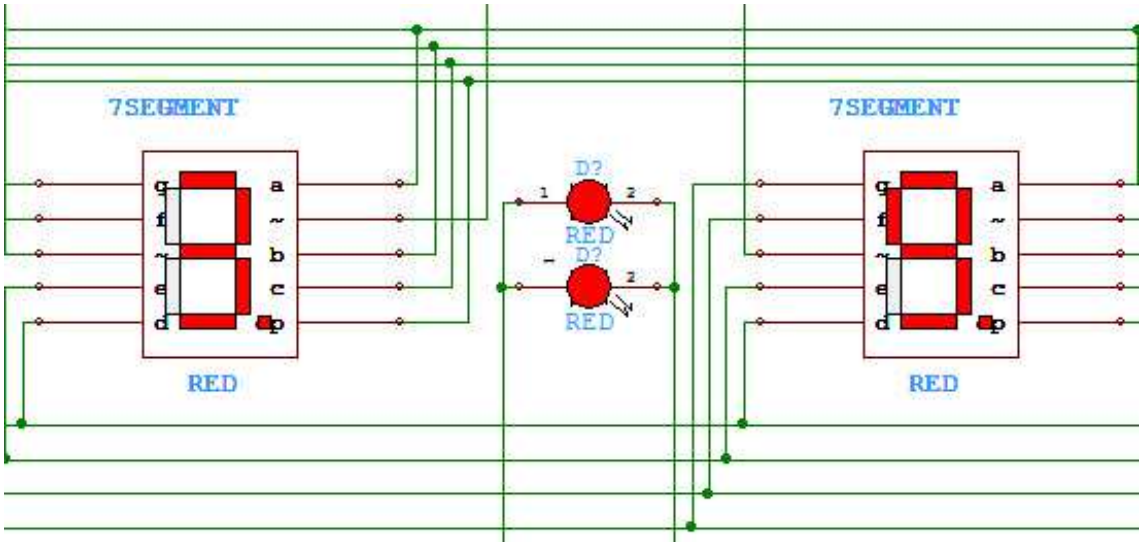
```
if Shape[i] <> nil then
      begin
        Shape[i].PenColor := clBlack;
        Shape[i].FillColor := LEDColor;
        Shape[i].BackgroundColor := $F0F0F0;
      end;
```

- *Takto lze nastavit jednotlivým tvarům, jakou barvou budou vyplněny.*

Překreslení součástky opět obstará metoda Draw.

```
procedure Draw(paper: TCanvas; Erase, Highlighted: Boolean);

for i := 0 to pocet do
  begin
   if Shape[i] <> nil then
    begin
     if Light[i] then Shape[i].Filled := 'F'
     else Shape[i].Filled := 'f';
     Shape[i].Draw(Paper, PosX, PosY, A, B, C, D, false, false);
    end;
  end;
```

- *Pole tvarů je procházeno, každý tvar nese parametr Filled, kde „F“ znamená vyplněný a „f“ nevyplněný. Nakonec je tvar metodou Draw nakreslen do schématu.*



Obr. 6 - Rozsvícené segmenty displeje

4.2 Programování vlastností součástek

Je-li hotov grafickým návrh součástky. Zbývá naprogramovat její funkčnost. Pro zařazení součástky do schématu a výpočet ustáleného stavu obvodu v zásadě stačí naprogramovat její proudové vlastnosti (funkce GetCurrent) a metodu Create. Například takto by se naprogramoval rezistor:

4.2.1 Rezistor:

Rezistor (odpor) má 2 piny, není závislý na polaritě a výpočet proudu určuje Ohmův zákon,
tedy vzorec: I = U/R,

kde: I = výsledný proud součástkou

U = napětí procházející součástkou o odporu R

Metoda Create:

- *pomocí inherited dědíme vlastnosti obecné součástky TPart*

```
constructor TResistor.Create(p: TPart);
   begin
      inherited;
```

- *dále je potřeba inicializovat všechny piny a načíst hodnotu odporu z pole TField, které obsahuje hodnoty zadané při návrhu součástky v programu KiCad, pokud by načtení selhalo, je defaultně přiřazena hodnota (v tomto případě 1kΩ)*

- *pomocí AddToInfoList('text') je vypsána poznámka do infolistu HyCiSim*

```
     InitializePins(2);
Resistor := DecodeResistorValue(TField(Fields.Items[1]).Text);
     if Resistor = 0 then
        begin
          Resistor := 1000;
          AddToInfoList('Error in part ' + Reference + ':
Resistivity value field is invalid, value is 1k now');
        end;
   end;
```

Funkce GetCurrent:

```
function TResistor.GetCurrent(pin: Integer): TFloat;
begin
   Result := 0;
```

- *piny odporu jsou dva, jeden je tedy v seznamu pinů pod indexem 0 a druhý 1*
- *vezme-li se napětí na jednom z pinů a odečte se od něj napětí na druhém pinu, zjistí se hodnota napětí na odporu / Rezistor (hodnota odporu) = proud*
- *v případě, že je dotazován proud na pinu 1,funkce vrátí výsledek s opačným znaménkem (v rámci podschématu je podle Kirchhoffových zákonů součet proudů 0)*

```
if (pin in [0 , 1]) and (PinList[0] <> nil) and (PinList[1]
<> nil) then
   begin
Result:=(PinList[1].GetVoltage-PinList[0].GetVoltage)/Resistor;
      if Pin = 1 then Result := - Result;
   end;
end;
```

4.2.2 LED dioda

Složitější součástkou je LED dioda. Výpočet proudu již nebude jednoduchá lineární funkce. Navíc bude nutné nadefinovat metodu pro rozsvícení diody pokud proud překročí určitou mez (metoda AfterChange) a metodu pro překreslení součástky (ReDraw).

Metoda Create:

- *opět 2 piny (0,1), proud diodou se vypočítá uvedou funkcí*

```
const
k1 = 1e-4; k2 = 3.96;


temp := (PinList[0].GetVoltage - PinList[1].GetVoltage);
if temp <= 0 then temp := 0
else temp := k1 * (Exp( k2 * temp / Voltage ) - 1);
if Pin = 1 then Result := temp else Result := - temp;
```

Metoda AfterChange:

Tato metoda se volá při změně hodnoty proudu na některé z pinů, v metodě create je nutno nastavit proměnnou OnChangeActive := true;, na základě výpočtu proudu je nastavena proměnná Light na true nebo false.

```
procedure TLEDDiode.AfterChange(pin: Integer);
var
temp : TFloat;
begin
  inherited;
  if (pin in [0, 1]) and (PinList[0] <> nil) and
(PinList[1] <> nil) then

begin
temp := (PinList[0].GetVoltage - PinList[1].GetVoltage);
if temp <= 0 then temp := 0
else
temp := k1 * (Exp( k2 * temp / Voltage ) - 1);

if (temp > 0.005) <> Light then
  begin
    Light := (temp > 0.005);
```

- *je-li Light true (dioda bude svítit) vyvolá se událost překreslení součástky*

```
if Light then SchemeSheduler.AddRedrawPartEvent(self);
```

- *jinak je událost zrušena*

```
else
  begin
    SchemeSheduler.CancelPartRequestEvent(self, 0);
```

Reálná dioda nezhasne hned, ale až po čase perzistence, tuto vlastnost simuluje událost AddPaerRequestEvent, kde rezistence znamená zpoždění v řádu milisekund

```
  SchemeSheduler.AddPartRequestEvent(Self, 0, Persistance);
 end;
end;
```

Dále je potřeba rozsvícení vykreslit na schéma. K tomu použijeme metody Draw a ReDraw.

Metoda Draw:

Tato metoda slouží k vykreslení součástky na plátno, bude potřeba i metodu ReDraw, vyvolávaná událostí AddRedrawPartEvent, k zamezení duplicity kódu Draw pouze volá ReDraw

```
procedure TLEDDiode.Draw(paper: TCanvas; Erase,
Highlighted: Boolean);
begin
  inherited;
  Redraw(Paper);
end;
```

Metoda ReDraw:

```
procedure TLEDDiode.ReDraw(paper: TCanvas);
var
d : Integer;
begin
  inherited;
```

- *d slouží k přepočtu souřadnic elipsy (tvar diody) na základě zvětšení ZOOM*

```
d := 70 div ZOOM;
```

- *má-li být LED dioda rozsvícená (Light = true), nastaví barvu štětce na LEDColor (ta je načtena v metodě Create z pole součástky TField) jinak na clSilver (nesvítí)*

```
if Light then Paper.Brush.Color := LEDColor else
  Paper.Brush.Color := clSilver;
  Paper.Pen.Color := clBlack;
  Paper.Pen.Width := 1;
```

- *zbývá už jen vykreslit požadovanou elipsu*

```
Paper.Ellipse(PosX div ZOOM - d, PosY div ZOOM - d, PosX
div ZOOM + d, PosY div ZOOM + d);
end;
```

5 Simulace mikrokontroléru PIC16F877A

Úvodem je vhodné říci, co si pod pojmem mikrokotrolér představit. Jedná se o složitý logický obvod, který je schopný vykonávat určitou řídící, kontrolní či matematickou funkci pomocí základních logický, aritmetických a datových instrukcí. Každá instrukce trvá určitou dobu, v závislosti na frekvenci mikroprocesoru a jeho instrukčním cyklu. Paměť pro data a instrukce může být buď společná (Von Neumannova architektura) nebo je rozdělena na datovou a programovou paměť (harvardská architektura). Harvardská architektura má navíc tu výhodu, že může být použita jiná šířka sběrnice pro data a jiná pro programovou paměť. Tím se dá výrazně zredukovat instrukční sada a zkrátit instrukční cyklus mikroprocesoru. Další součástí jsou vstupně/výstupní obvody (porty), kterými lze buď číst hodnotu z externích obvodů (vstup) nebo nastavit hodnotu na pinu mikrokontroléru (výstup). Typicky obsahuje mikrokontrolér další integrované obvody, které se nazývají integrované periferie. Jsou jimi čítače/časovače, A/D převodníky, porty pro připojení datových sběrnic (RS-232, USART, I^2C a jiných), komparátory, modulační obvody atd.

5.1 Charakteristika mikrokontroléru PIC16F877A

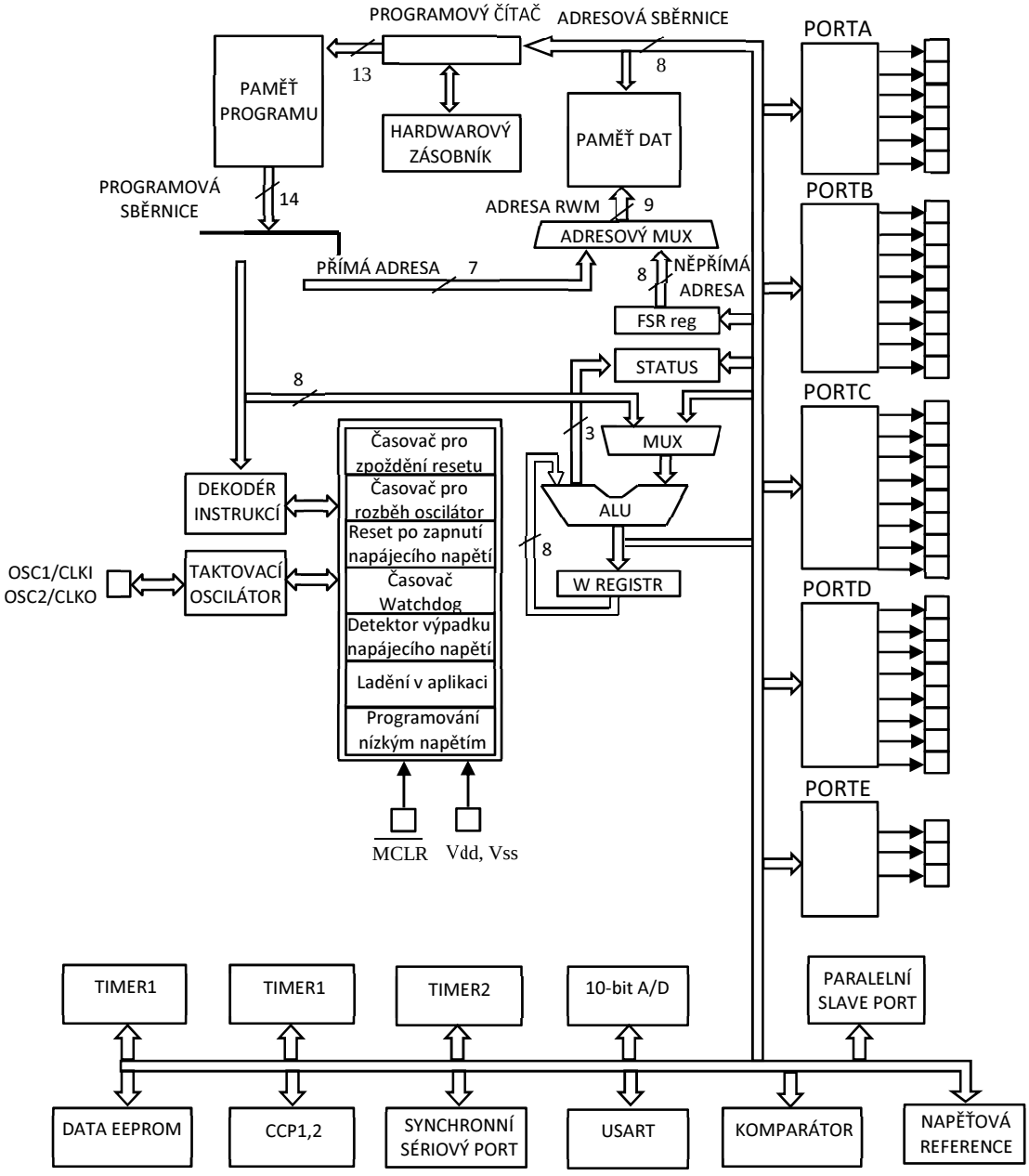
PIC16F877A je osmibitový mikrokontrolér harvardské architektury. Na bázi RISC architektury všechny instrukce trvají jeden instrukční cyklus (vyjma instrukcí větvení programu), tedy frekvence oscilátoru - Fosc/4 a zabírají jedno místo v paměti programu. Vyrábí se v pouzdře se 44 vývody nebo 40 vývody.

Tabulka 1 - Varianty mikrokontroléru PIC16F87XA

Parametr	PIC16F873A	PIC16F874A	PIC16F876A	PIC16F877A
Taktovací frekvence	0 – 20 MHz	0 – 20 MHz	0 – 20 MHz	0 – 20 MHz
Obvody Resetu	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
Velikost paměti programu Flash (14-bitový slov)	4K	4K	8K	8K
Velikost paměti dat RWM (bajtů)	192	192	368	368
Velikost paměti dat EEPROM (bajtů)	128	128	256	256
Přerušovací zdroje	14	15	14	15
Vstupně výstupní brány	Port A, B, C	Port A, B, C, D, E	Port A, B, C	Port A, B, C, D, E
Čítače/časovače	3	3	3	3
Moduly Capture/Compare/PWM	2	2	2	2
Sériová komunikace	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Paralelní komunikace	—	PSP	—	PSP
10-bitový AD převodník	5 vstupů	8 vstupů	5 vstupů	8 vstupů
Analogový komparátor	2	2	2	2
Instrukční soubor	35 instrukcí	35 instrukcí	35 instrukcí	35 instrukcí
Pouzdra	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN	40-pin PDIP 44-pin PLCC 44-pin TQFP 44-pin QFN	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN	40-pin PDIP 44-pin PLCC 44-pin TQFP 44-pin QFN

5.2 Blokové schéma

Na obrázku 7 je vidět blokové schéma mikrokontroléru PIC16F877A. Vpravo jsou vstupně/výstupní porty. V horní části paměť programu o velikosti slova 14 bitů, kterou lze adresovat pomocí 13-bitového programového čítače. Dále zápisníková paměť, která obsahuje řídící a datové registry. Přistupovat k ní lze přímou nebo nepřímou adresací. Uprostřed je aritmeticko-logická jednotka ALU, z které lze zapisovat buď do pracovního registru W nebo do zápisníkové paměti. Příznaky z výsledku aritmeticko-logických operací jsou nastaveny v registru STATUS, jehož další funkce je volba banky zápisníkové paměti při přímé adresaci, při nepřímé adresaci obsahuje 9. bit adresy (dolních 8 bitů je uloženo v registru na adrese obsažené ve FSR registru). V dolní části schématu jsou jednotlivé integrované periferie.



Obr. 7 - Blokové schéma mikrokontroléru PIC16F877A

5.3 Návrh paměťové struktury

Návrh paměti programu je velice jednoduchý z toho důvodu, že lze adresovat v celém rozsahu pomocí 13-bitového programového čítače. Stačí tedy vytvořit pole o velikosti 8192 slov. Přičemž jedno slovo má velikost 14 bitů a jeho hodnota po restartu jsou samé jedničky – tedy 0x3FFF.

Mnohem složitější je realizace přístupu k datové paměti. Ta je sice menšího rozsahu (512 slov) a velikost jednoho datového slova je 1 byte, ale problém nastává v její adresaci. Paměť je rozdělena na 4 banky po 128 buňkách. Pro její adresaci existují dva způsoby: a) přímou adresou

b) nepřímou adresou

Přímá adresa se skládá ze 7 bitů uložených v některém z dostupných registrů zápisníkové paměti a z bitů RP1 a RP0 v registru STATUS, kterými se volí číslo banky. K nepřímé adresaci slouží registr INDF, který není skutečným registrem. Ve skutečnosti se pracuje s registrem, jehož adresa je uložena v registru FSR (ukazatel) a 9. bit adresy získáme ze 7. bitu registru STATUS.

Dalším aspektem při vytváření metod přístupu k zápisníkové paměti je, že některé významné registry jsou obsaženy ve více bankách paměti (viz. Tabulka 2 - Struktura zápisníkové paměti).

5.3.1 Metody pro práci s pamětí:

5.3.1.1 Paměť programu:

- lze nadefinovat jako jednoduché pole:

```
CDATA : array[0..8191] of word;
```

Pro práci s tímto polem (pamětí) budou sloužit jednoduché čtecí a zapisovací funkce:

```
function GetCDATA(addr: Word): word;
begin
  GetCDATA := CDATA[addr];
end;
```

- *metoda GetCDATA vrátí datové slovo v paměti programu na cílové adrese addr*

```
procedure SetCDATA(addr: word; data: word);
begin
  CDATA[addr] := data;
end;
```

- *SetCDATA uloží nová data na adresu addr v paměti programu*

5.3.1.2 Paměť dat (zápisníková paměť):

V této paměti jsou umístěny speciálně funkční registry SFR, jejich adresy jsou v programu nadefinovány jako konstanty. V případě, že se daný SFR objevuje ve více buňkách, uložíme těchto více adres do konstantního pole, například:

```
SFR_TMR0: array [0..1] of word = ($01,$101);
SFR_INDF: array [0..3] of word = ($00,$80,$100,$180);
```

Paměť je nadefinována jako:

```
IDATA : array[0..511] of byte;
```

Přístupové metody:

```
procedure SetIDATA(addr: Word; data: Byte);

IDATA[addr]:=data;
if ((addr = SFR_TMR0[0]) or (addr = SFR_TMR0[1])) then
    begin
      IDATA[SFR_TMR0[0]] := data;
      IDATA[SFR_TMR0[1]] := data;
    end;

if ((addr = SFR_INDF[0]) or (addr = SFR_INDF[1])...  then
    begin
      a := GetSFR(SFR_FSR[0]); //8 dolních bitů adresy
      b := (GetSFR(SFR_STATUS[0]) and $80) * 2; //9. bit adresy
      IDATA[a+b] := data;
    end;
```

SetIDATA slouží pro zápis do paměti. Ať už přímou adresací a nebo nepřímou adresací pomocí registru INDF. Pokud je některý z registrů v paměti přístupný na více místech, je nutné na všechny tyto adresy rozkopírovat jeho hodnotu viz. ukázka kódu.

Pro čtení stačí jednodušší funkce GetIDATA, která bude ještě v další části rozšířena pro práci s porty.

```
Function GetIDATA(addr: word;): byte;
begin
  GetIDATA:=IDATA[addr];
end;
```

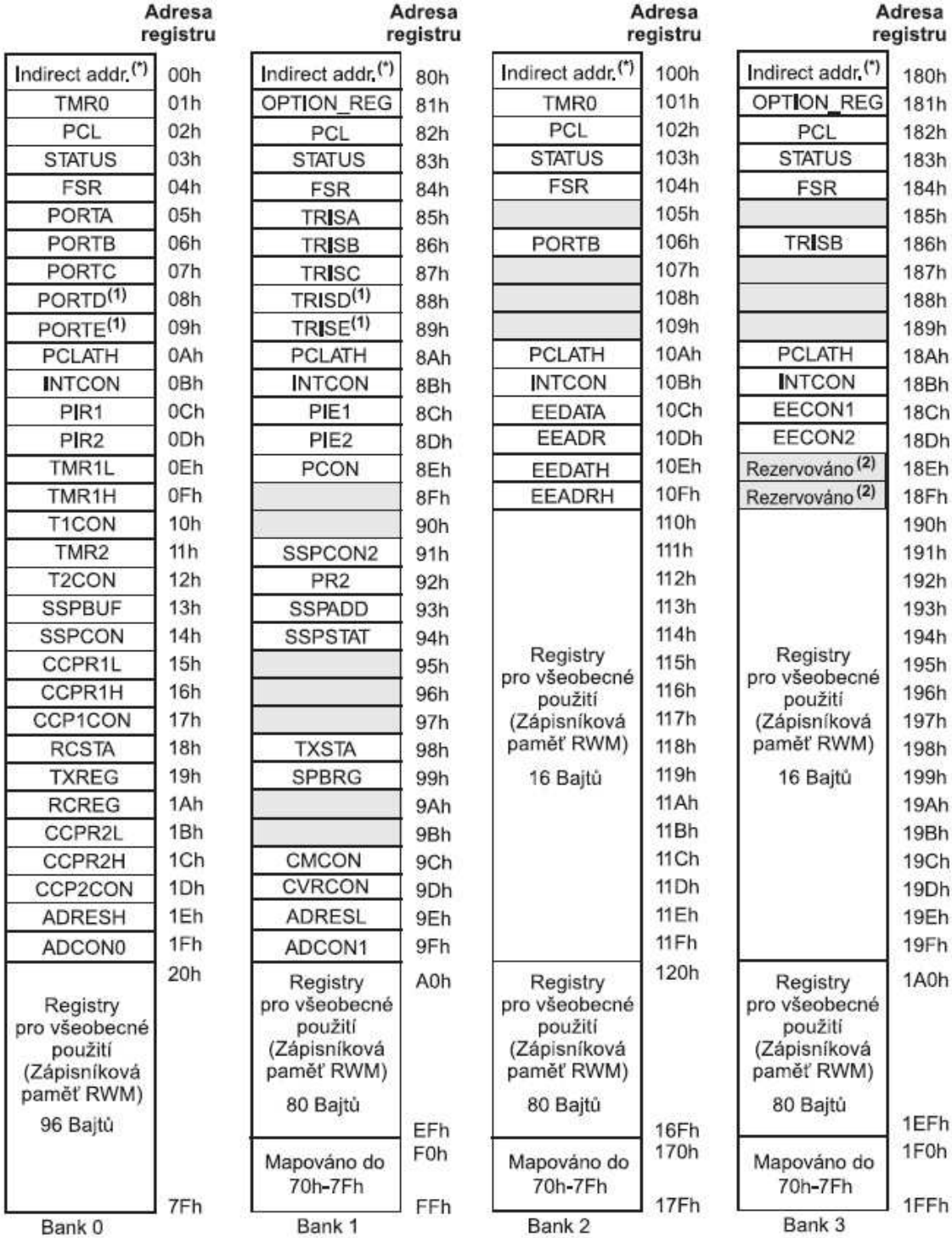
Po restartu procesoru se nastaví paměť IDATA na počáteční hodnotu. Většina buněk má nulovou hodnotu, ale SFR mohou mít jiné nastavení. Pro tento účel slouží metody ClearMemory a ResetProcesor.

```
procedure ResetProcesor;
begin
  SetIDATA(SFR_STATUS[0],$18,true);
  SetIDATA(SFR_TRISA,$3f,true);
```

Metoda ClearMemory vymaže paměť dat a programu v celém rozsahu (například při inicializaci procesoru nebo po resetu).

```
procedure ClearMemory;
begin
  for i := 0 to 511 do IDATA[i] := 0;
  for i := 0 to 8191 do CDATA[i] := $3FFF;
end;
```

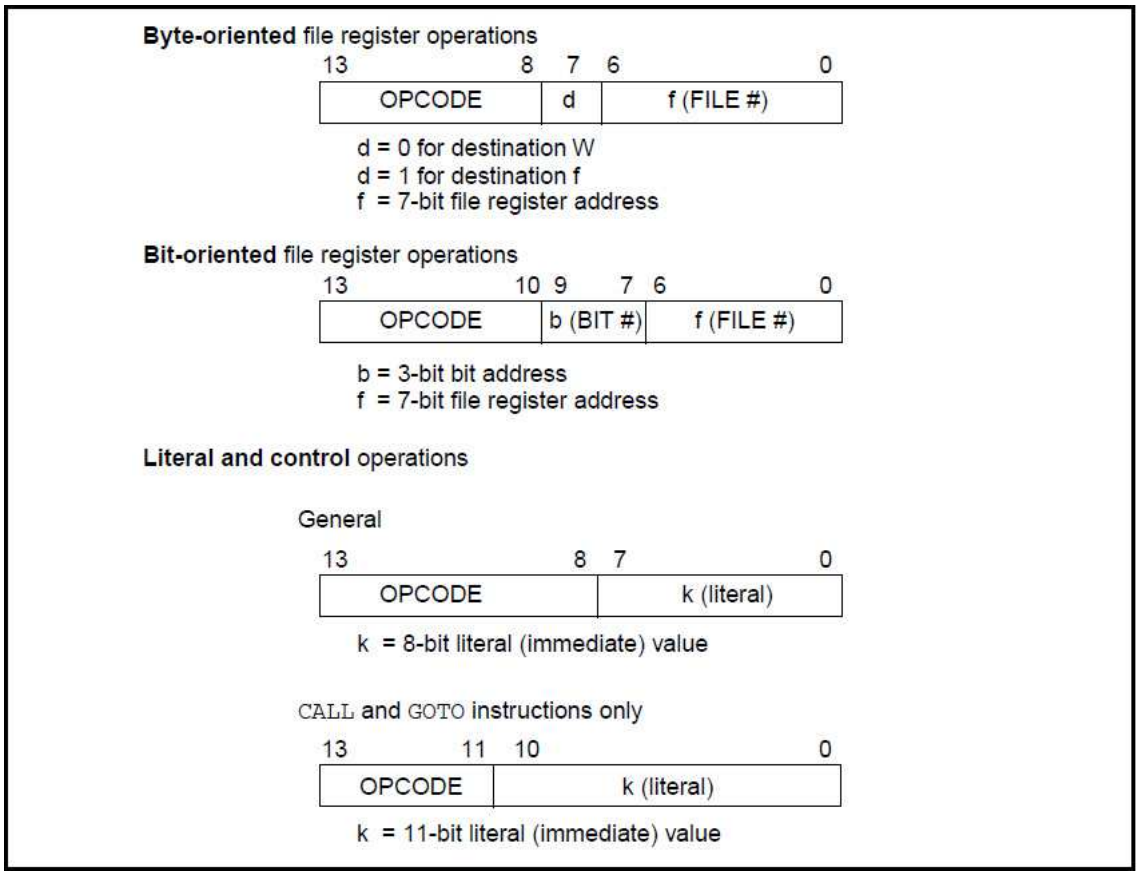
Tabulka 2 - Struktura zápisníkové paměti



5.4 Instrukce mikrokontroléru

5.4.1 Instrukční soubor

Veškeré instrukce jsou popsány v instrukčním souboru. Mikrokontrolér PIC16F877A používá 35 instrukcí. Dělí se na bytově-orientované, bitově-orientované a instrukce pracující s konstantou. Jednotlivé druhy instrukcí mají jiný formát, pod kterým jsou uloženy v paměti programu (viz. obrázek 8 – Formát instrukcí). Všechny instrukce mají délku 14 bitů a trvají jeden procesorový cyklus (kromě instrukcí větvení programu).



Obr. 8 - Formát instrukcí

OPCODE – operační kód instrukce, na jeho základě mikroprocesor rozpozná, kterou instrukci bude vykonávat

d - místo pro uložení výsledku

- d = 0, výsledek uložen do pracovního registru W

- d = 1, výsledek uložen do registru na adrese f

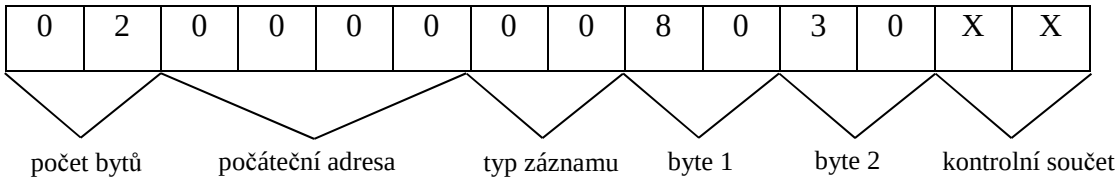
b - pozice bitu slova uloženého na adrese f (3 bity, rozsah 0-7)

k - konstanta (8 bitů, u instrukcí CALL a GOTO 11 bitů)

5.4.2 Načítání instrukcí z HEX souboru

Firma Microchip Technology má své vlastní návrhové prostředí MPLAB IDE. Kód v něm napsaný je uložen do HEX souboru.

5.4.2.1 HEX formát:



5.4.2.2 Příklad instrukce uložené v HEX souboru:

:0400000080308407C1

04 – na řádku jsou 4 byty s informacemi o instrukcích (každá instrukce zabírá 2 byty)

0000 – startovací adresa v paměti 0h

00 – typ záznamu (datový záznam)

8030 – 1. instrukce (pořadí je prohozené, OPCODE = 30, adresa nebo konstanta = 80)

8040 – 2. Instrukce

5.4.2.3 Dekódování instrukce:

Instrukci lze rozpoznat na základě OPCODE. Do paměti programu se neukládá ve tvaru 8030, jak je vidět v příkladu nahoře, ale 3080. Kdy 30 = OPCODE a 80 = adresa registru nebo konstanta. Následující kód patří instrukci MOVWF a její OPCODE je 1100XX. Převedeno do šestnáctkové soustavy 30h. Znamená to tedy, že 8030 v HEX souboru značí instrukci MOVWF 0x80.

11	00xx	kkkk	kkkk

Pro vykonávání instrukcí je tedy potřeba vytvořit metody:

LoadHex - načítání dat z HEX souboru a jejich ukládání do paměti programu

DecodeInstruction – určení konkrétní instrukce na základě jejího OPCODE

ExecuteInstruction – vykonání instrukce

Metoda LoadHex má na vstupu řádek z HEX souboru. Podle Intel HEX formátu zjistí počet bytů (len) a počáteční adresu (adr), která je vydělená dvěma, jelikož se instrukce skládá ze dvou bytů. Do proměnné „b“ je načtena zakódovaná instrukce.

```
procedure LoadHex(line:string);
  begin
    if Length(line)>0 then if line[1]=':' then
      begin
        len:=16*DeHex(line[2])+DeHex(line[3]);
        adr:=16*DeHex(line[4])+DeHex(line[5]);
        adr:=(256*adr+16*DeHex(line[6])+DeHex(line[7]))div 2;

        if len>2 then
        for i:=0 to ((len  div 2)-1) do
          begin
b:=16*DeHex(line[4*i+10])+DeHex(line[4*i+11])
   +4096*DeHex(line[4*i+12])+256*DeHex(line[4*i+13]);
            SetCDATA(adr+i,b);
          end;
      end;
  end;
```

DecodeInstuction prochází instrukční soubor. Vrací název instrukce a její parametry ve formě textového řetězce. Jedinou vstupní informací je index na adresu paměti programu.

ExecuteInstruction má nastavenu konstantu „execute“ a spustí navíc oproti metodě DedoceInstruction vykonání dané instrukce.

Instrukční soubor obsahuje výkonný kód všech instrukcí. Například „MOVF f, d“, která uloží data na adrese f do pracovního registru nebo zpět do f z důvodů testování na nulový příznak.

```
if (CDATA[index] and $3F00) = $0800 then
   begin
      {$ifdef execute}
         begin
            p:=GetIDATA(SFR_STATUS[0],true);
  b1 := 32*((p div 32) and $03)+(CDATA[index] and $7F);  //adr f
            b2 := CDATA[index] and $80;  //d
            w := GetIDATA(b1, true);
  if w = 0 then SetIDATA(SFR_STATUS[0], p or $04, true); //Z
            if b2 = $80 then
               SetIDATA(b1, w, true)
            else
               etIDATA(SFR_W, w, true);
         end;
      {$endif}
   end;
```

Do „p“ je uložen obsah registru STATUS, „b1“ nese adresu registru „f“ a v proměnné „w“ je jeho hodnota, podle hodnoty „b2“ bude „w“ uložen buď do pracovního registru nebo zpět do registru „f“. Je-li „w“ nulové, je nastaven bit Z v registru STATUS.

5.5 Takt mikroprocesoru

Rychlost procesoru není dána pouze frekvencí vnitřního oscilátoru, ale záleží i na délce instrukčního cyklu. Instrukční cyklus je posloupnost kroků, která je potřeba pro vykonání jedné instrukce. Počet těchto kroků odpovídá počtu taktů procesoru. Některé procesory mají instrukce s různým počtem instrukčních cyklů. PIC16F877A má tu výhodu, že všechny instrukce trvají stejný počet instrukčních cyklů, a to jeden (kromě instrukcí větvení programu, ty zabírají cykly 2). Počet taktů, který zabere jeden cyklus je čtyři. Jedna instrukce tedy bude vykonána za:

$$t = \frac{1}{(Fosc \div 4)}$$

t – čas v sekundách
Fosc – frekvence oscilátoru v hertzích

Poznámka: V případě PIC 16F877A je Fosc 20 MHz, takže jedna instrukce trvá 0,2 μS.

Konkrétní kroky instrukčního cyklu jsou:

1. získání OPCODE instrukce z instrukčního slova (anglicky „Fetch cycle“)
2. dekódování instrukce pro další krok
3. zjištění o jaký typ instrukce se jedná, jedná-li se o instrukci přístupu do paměti a přímou adresaci, neprovede nic, pokud jde o nepřímou adresaci přečte data z určené adresy, instrukce registrů a I/O portů jsou v tomto kroku provedeny
4. posledním krokem je vykonání instrukce: matematické či logické operace nad získanými daty, přesun dat mezi I/O porty a procesorem

V simulátoru HyCiSim je pro napodobení přesného řízení chodu mikroprocesoru použita metoda OnMyRequest. V jejím těle je definováno, co všechno procesor během jednoho instrukčního cyklu provede. Mezi těmito úkoly je vykonání instrukce, která je na řadě, obsluha čítače/časovače, obsluha přerušení a další specifické úlohy, jako je uložení předchozích stavů portů atd. Na konci OnMyRequest zavolá metodu AddPartRequestEvent z knihovny Scheduler, která má na svém vstupu součástku vyvolávající obsluhu (TPart), ID obsluhy a časové zpoždění. Nastaví-li se zpoždění na Fosc/4, bude se metoda OnMyRequest opakovat v přesných intervalech, jako je tomu u instrukčního cyklu procesoru.

```
procedure OnMyRequest(ID: Integer);
      .
      ExecuteInstruction(PC,bytes,cycles);
      HandleTimers; //obsluha čítačů/časovačů
      .
      .
      .
      //uložení hodnot portů v předchozím instrukčním cyklu
      Old_PORTA := GetSFR(SFR_PORTA);
      Old_PORTC := GetSFR(SFR_PORTC);
     end;
SchemeSheduler.AddPartRequestEvent(self, 0, 4 / Fosc);

end;
```

6 Vstupně/výstupní porty

Mikrokontroléry PIC16F877A má 5 vstupně/výstupních portů. Jejich vývody se dají použít jako digitální vstupně/výstupní piny. Některý z nich slouží pro činnost periferií, ve většině případů platí, že pokud je používán periferií, nelze pin zároveň využít jako digitální vývod. Každý port má svůj pracovní záchytný registr, jeho čtením lze zjistit stav vývodů, který je určován vnějším prostředím. Zápis do tohoto registru se nastaví data, která bude mikroprocesor udržovat na svých vývodech. Každá zapisovací operace je typu „read-modify-write“ (čti-modifikuj-zapiš). To znamená, že před zápisem na port je celý port načten, změněn a až následovně se zapíší data do pracovního záchytného registru.

Registr TRIS je konfigurační registr portu. Zápis log. 1 do příslušného bitu registru TRIS nastaví daný pin jako vstup. Nulování bitu v registru TRIS nastaví daný pin jako výstup.

6.1 Metody pro práci s porty

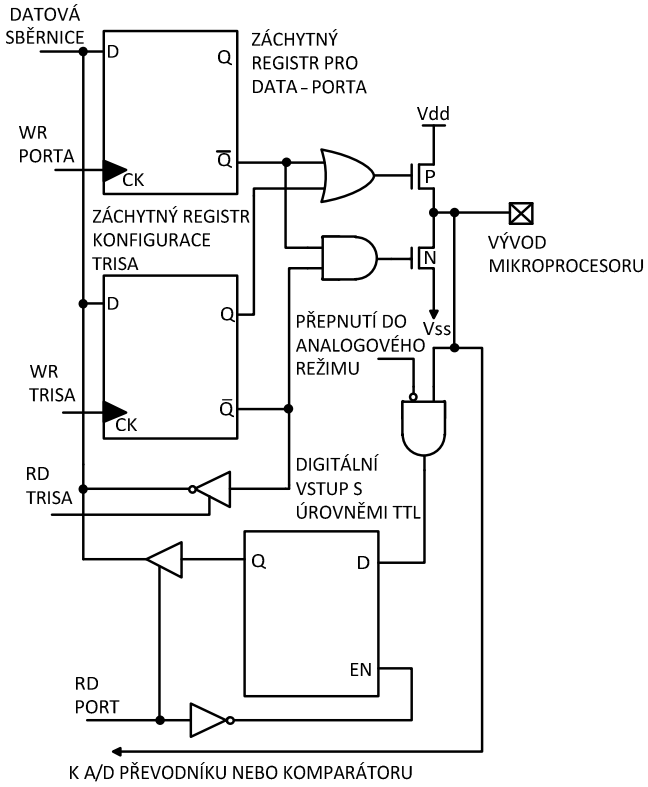
Výpočet proudu na jednotlivých pinech provádí metoda GetCurrent. Následující ukázka kódu počítá proud na jednotlivých pinech portu A. Podle registru TRISA je příslušný pin nastaven jako vstupní nebo výstupní. Je-li pin nastaven jako výstupní (TRIS<pin> = 0), zjistí se vymaskováním úroveň, která má být na výstupu pinu. Výpočet proudu vyplývá z obrázku 9 - Blokové schéma portu A.

```
function GetCurrent(pin: Integer): TFloat;
begin

//PORT A
  if (pin in [1..6]) and (PinList[pin] <> nil) then
    begin
      if ((GetSFR(SFR_TRISA) and Weight[pin - 1]) = $00) then
        begin
          if (OutPort[0] and Weight[pin - 1] > 0) then
            Result := (5 - PinList[pin].GetVoltage) / 200
          else
            Result := (0 - PinList[pin].GetVoltage) / 200;
        end
      else Result := (5 - PinList[pin].GetVoltage) /52200000;
    end;
```

Pokud je pin nastaven jako výstupní, je příslušným bit registru TRISA nulový. Úroveň na výstupu je určována příslušným bitem záchytného registru PORTA. Nastavením příslušného bitu záchytného registru PORTA je tranzistor typu P otevřen a tranzistor N uzavřen. Na výstupu bude proud maximálně 25 mA (hodnota proudu je vypočítána na základě datasheetu mikroprocesoru PIC16F877A, kde je definováno napájecí napětí 5V a vnitřní odpor 200Ω), je-li bit registru PORTA nulový bude tranzistor P uzavřen a tranzistor typu N otevřen, na výstupu je tedy nulové napětí. Hodnota 1 v registru TRISA nastaví příslušný vývod jako vstup. Oba tranzistory jsou v tomto případě uzavřeny a hodnota vnějšího obvodu lze vyčíst na datové sběrnici při signálu RD PORT („read port“). Změna vnějším obvodem vyvolá metodu AfterChange.

```
Procedure AfterChange(pin: Integer);
begin
//PORT A
if ((GetSFR(SFR_TRISA) and Weight[pin - 1]) = Weight[pin - 1])
then
  begin
  if (pin in [1..6]) and (PinList[pin] <> nil) then
    begin

if (InPort[0] and Weight[pin - 1] > 0) and
    PinList[pin].GetVoltage < OffInput) then
begin
  InPort[0] := InPort[0] and (not Weight[pin - 1]);
  SetSFR(SFR_PORTA, GetSFR(SFR_PORTA) and (not Weight[pin-1]));
end
else
  if (InPort[0] and Weight[pin - 1] = 0) and
     (PinList[pin].GetVoltage > OnInput) then
    begin
     InPort[0] := InPort[0] or Weight[pin - 1];
     SetSFR(SFR_PORTA, GetSFR(SFR_PORTA) or Weight[pin - 1]);
    end;
end;
```

- *Metoda AfterChange nejprve zkontroluje, je-li pin nastaven jako vstupní. Pokud ano, je porovnána logická úroveň pinu před zavoláním metody AfterChange a logická úroveň na vnějším obvodu. Při změně je přepsán příslušný bit registru PORTA a pomocné pole InPort s hodnotami výstupních pinů.*



Obr. 9 - Blokové schéma portu A

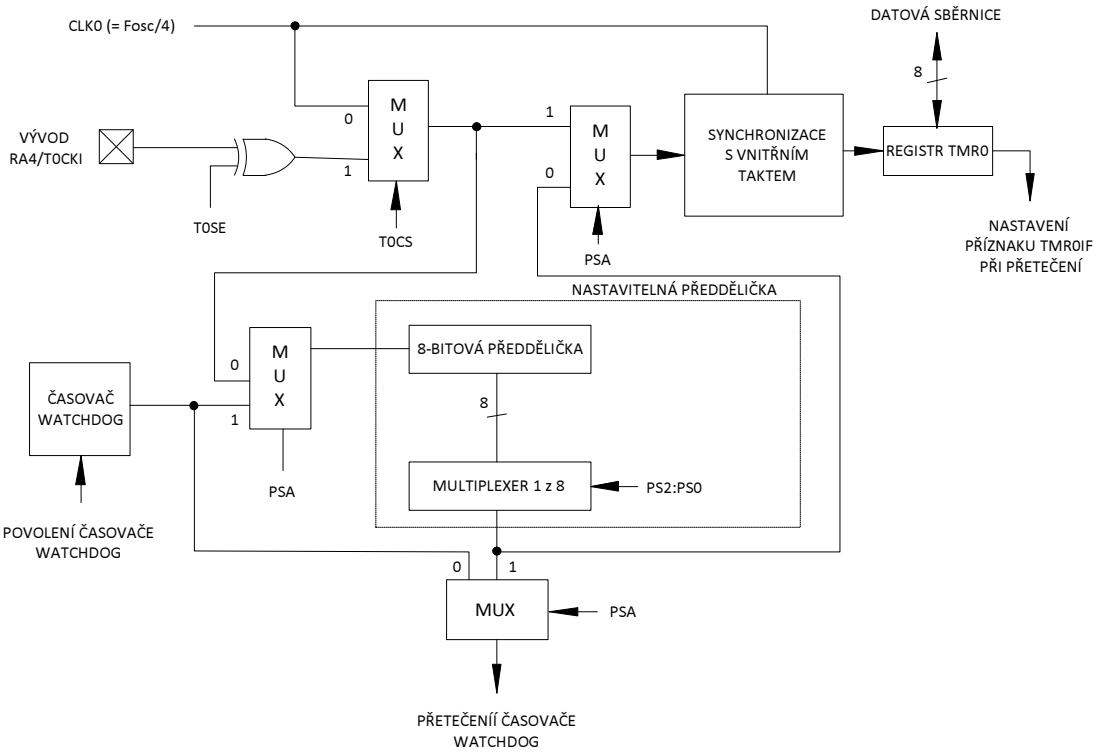
7 Obvody čítače/časovače

Mikrokontrolér PIC16F877A má celkem tři čítače/časovače (Timer0, Timer1, Timer2). Pokud je obvod nastaven jako čítač, inkrementuje svůj pracovní registr, který zároveň určuje šířku čítače/časovače, na základě změny na vstupním pinu k tomu určenému. Tato změna může nastat přechodem ze stavu log. 0 do stavu log. 1 (náběžná hrana pulzu) nebo ze stavu log. 1 do stavu log. 0 (sestupná hrana). Je-li pracovní registr čítače naplněn, dojde v dalším taktu k jeho přetečení a nastavením příznaku přetečení. Pokud je povoleno přerušení od čítače/časovače, dojde k jeho obsluze, ve které většinou musí být vynulován softwarově příznak přerušení.

Ve funkci časovače dochází v každém taktu procesoru k inkrementaci pracovního registru (není-li zařazena předdělička), jeho přetečení opět nastaví příznak přetečení čítače/časovače. Předdělička je další čítač, který může být předřazen nebo zařazen za čítač/časovač. Obvod předděličky slouží k prodloužení periody inkrementace pracovního registru čítače/časovače (k inkrementaci dojde až po přetečení předděličky). Tento poměr lze u programovatelné předděličky volit.

7.1 Timer0

Modul Timer0 slouží jako 8-bitový čítač/časovač s 8-bitovou programovatelnou předděličkou. Lze řídit buď vnitřním taktem mikroprocesoru nebo vnějším hodinovým pulzem přivedeným na pin RA4/TOCKI, který je synchronizován druhým a čtvrtým strojovým taktem instrukčního cyklu mikroprocesoru. Logická úroveň tohoto vnějšího oscilátoru tedy musí trvat minimálně dva strojové takty mikroprocesoru. Volba zdroje hodinového signálu se provádí v 5. bitu T0CS registru OPTION (1 – vnější zdroj, 0 - vnitřní takt Fosc/4). TMR0 je 8-bitový pracovní registr přístupný pro čtení i zápis, přetečení z hodnoty FFh na 00h nastaví příznak TMR0IF v registru INTCON<2>. Přerušení od Timer0 lze povolit nastavením bitu TMR0IE v registru INTCON. Dělící poměr předděličky nastavují bity PS0-PS2 registru OPTION (nutno nastavit i bit PSA pro přiřazení k obvodu Timer0) a lze nastavit plynule od 1:2 do 1:256.



Obr. 10 - Modul čítače/časovače Timer0

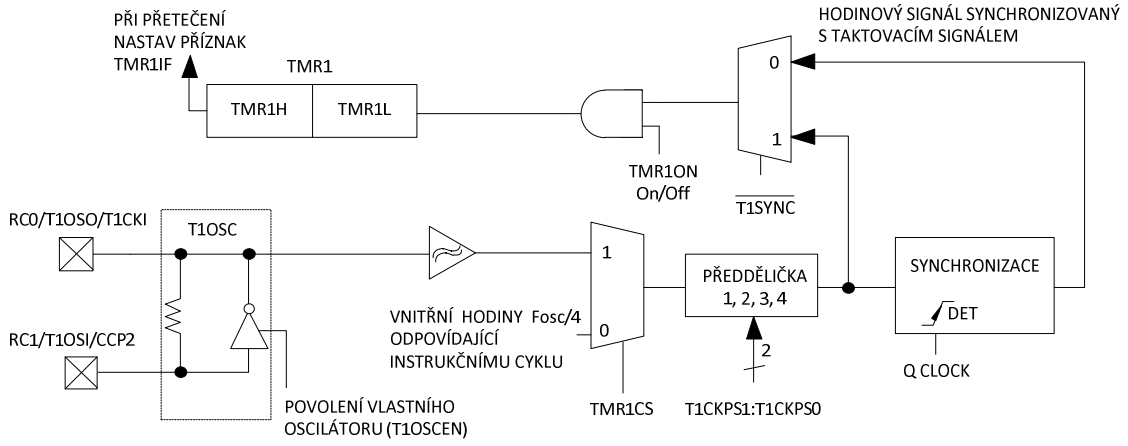
7.2 Timer1

Modul Timer1 je 16-bitový čítač/časovač realizovaný registry TMR1H a TMR1L. Oba registry lze číst i do nich zapisovat. Zdroje hodinového signálu se volí bitem TMR1CS (1 – vnější signál na pinu RC0, 0 – vnitřní signál Fosc/4). V režimu čítače může Timer1 pracovat synchronně nebo asynchronně s instrukčním cyklem mikroprocesoru na základě nastavení bitu T1SYNC. V synchronním režimu je čítač inkrementován s každou náběžnou hranou na pinu RC1 (pokud je nastaven bit OSCEN) nebo RC0 (bit OSCEN = 0) a nastavení registru TRISC je ignorováno. V asynchronním režimu je čítač inkrementován nezávisle na instrukčním cyklu mikroprocesoru.

V režimu časovače je Timer1 inkrementován v každém instrukčním cyklu (pokud není zařazena předdělička). Volba dělící poměru předděličky se provádí bity T1CKPS1:T1CKPS0 (poměr 1:1 až 1:8). Přetečení Timer1 nastaví příznak TMR1IF v registru PIR1, povolení přerušení od Timer1 nastavuje bit TMR1IE v registru PIE1.

Tabulka 3 - Registr T1CON

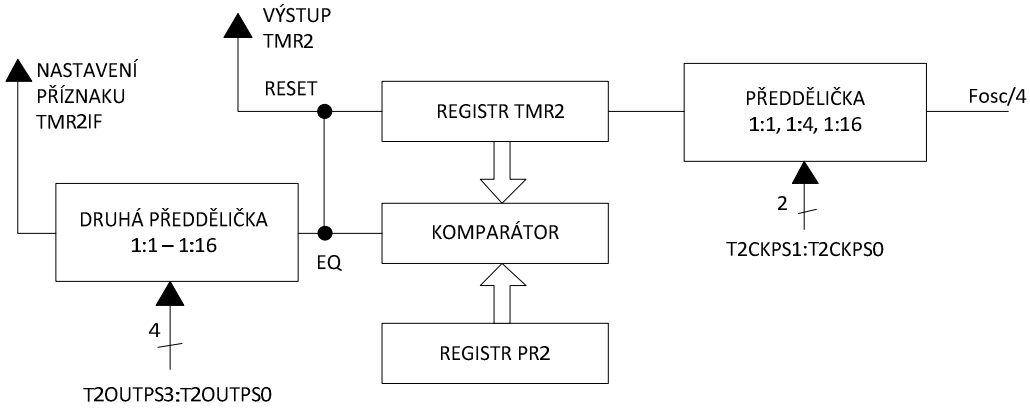
Adresa	Název	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Hodnota po resetu POR a BOR	Hodnota po ostatních druzích resetu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	--00 0000	--uu uuuu



Obr. 11 - Modul čítače/časovače Timer1

7.3 Timer2

Modul Timer2 je 8-bitový čítač/časovač, ke kterému lze přiřadit dvě 8-bitové předděličky. Zapnutí čítače/časovače Timer2 se provede nastavením bitu TMR2ON. Pracovní registr TMR2 lze libovolně číst i zapisovat. Pokud není zařazena předdělička, dochází k inkrementaci registru TMR2 do hodnoty uložené v registru periody PR2 v každém instrukčním cyklu. Po restartu je hodnota PR2 FFh. Poměr dělícího poměru předřazené předdělicky se volí bity T2SKPS1:T2CKPS0 (poměr 1:1, 1:4, 1:16), poměr předděličky zařazené za čítačem bity TOUTPS3:TOUTPS0.



Obr. 12 - Modul čítače/časovače Timer2

Tabulka 4 - Registr T2CON

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 **nepoužit:** při čtení '0'

bit 6-3 **TOUTPS3:TOUTPS0:** Volba dělicího poměru druhé předděličky zařazené za TMR2
0000 = dělicí poměr 1:1
0001 = dělicí poměr 1:2
0010 = dělicí poměr 1:3
•
•
•
1111 = dělicí poměr 1:16

bit 2 **TMR2ON:** Zapnutí modulu čítače/časovače Timer2
1 = Timer2 je zapnut
0 = Timer2 je vypnutý

bit 1-0 **T2SKPS1:T2CKPS0:** Volba dělicího poměru předděličky zařazené před TMR2
00 = dělicí poměr 1:1
01 = dělicí poměr 1:4
1x = dělicí poměr 1:16

7.3.1 Ukázka kódu čítače/časovače Timer0 v simulátoru HyCiSim

```
if (GetSFR(SFR_OPTION_REG[0]) and $20 = 0) then
```

- *je-li timer0 nastaven jako časovač*

```
  begin
    if (GetSFR(SFR_OPTION_REG[0]) and $08 = 0) then
```

- *je-li předdělička přiřazena obvodu timer0 zvýší se hodnota předděličky o 1 a zavolá se metoda pro obsluhu předděličky PrescalerT0*

```
      begin
        inc(pdd0);
        PrescalerT0(pdd0);
      end
    else
```

Metoda Prescaler:

- *nastavení prvních tří bitů PS0-PS3 v registru OPTION určuje poměr předděličky, dojde-li k přetečení přičte jedničku k registru TMR0*

```
procedure PrescalerT0(pdd: Byte);
begin
  case ((GetSFR(SFR_OPTION_REG[0]) and $07)) of
                0: if pdd0 = 2 then
                  begin

SetIDATA(SFR_TMR0[0],GetSFR(SFR_TMR0[0])+1,true);
                    pdd0 := 0;
                  end;
                .
                .
                .
                7: if pdd0 = 256 then
                  begin

SetIDATA(SFR_TMR0[0],GetSFR(SFR_TMR0[0])+1,true);
                    pdd0 := 0;
                  end;
              end;
if GetSFR(SFR_TMR0[0]) = $00 then
```

- *přetekl-li zároveň i registr TMR0, nastaví se příznak přetečení TMR0IF*

```
  SetIDATA(SFR_INTCON[0],GetSFR(SFR_INTCON[0]) or $04,true);
end;
```

- *pokud není zařazena předdělička, inkrementuje se registr TMR0 a přeteče-li z hodny FFh na 00h nastaví se příznak TMR0IF*

```
begin
  SetIDATA(SFR_TMR0[0],GetSFR(SFR_TMR0[0])+1,true);
  if GetSFR(SFR_TMR0[0]) = $00 then
    SetIDATA(SFR_INTCON[0],GetSFR(SFR_INTCON[0]) or $04,true);
end;
end;
```

- *je-li timer0 nastaven jako čítač*

```
if (GetSFR(SFR_OPTION_REG[0]) and $20 <> 0) then
  begin
```

- *je-li nastavena předdělička*

```
if (GetSFR(SFR_OPTION_REG[0]) and $10 = 0) then
```

- *tato podmínka kontroluje, jestli došlo k náběžné hraně na pinu T0CKI (v metodě OnMyRequest je vždy uložena hodnota záchytného registru portu A v minulém taktu do proměnné Old_PORTA)*

```
  if (((GetSFR(SFR_PORTA) and $10) = $10) and (Old_PORTA and $10
= 0)) then
        begin
```

- *opět dojde k obsluze předděličky, případně k přímé inkrementaci registru TMR0*

```
          if (GetSFR(SFR_OPTION_REG[0]) and $08 = 0) then
            begin
              inc(pdd0);
              PrescalerT0(pdd0);
            end
          else
            begin
             SetIDATA(SFR_TMR0[0],(GetSFR(SFR_TMR0[0])+1),true);
             if GetSFR(SFR_TMR0[0]) = 0 then
      SetIDATA(SFR_INTCON[0],GetSFR(SFR_INTCON[0]) or $04,true);
                end;
            end;
```

- *dále by následoval stejný kód pro sestupnou hranu na pinu T0CKI*

8 Obsluha přerušení

Mikrokontrolér PIC16F877A je vybaven celkem 15 zdroji přerušení. Jedná se o speciální příznaky vyvolané obvodem periferie. V základu se dělí na vnější přerušení (jedná se o přerušení periferie připojené zvenčí mikroprocesoru na jeho piny) a vnitřní přerušení (integrované periferie např. čítače/časovače). Těchto příznaků se dá využít například tehdy, pokud procesor využívá nějakou pomalou periferii, místo čekání může zpracovávat jinou úlohu a v momentě, kdy jsou data z periferie připravena, dostane signál v podobě příznaku přerušení. V takovém případě je na vrchol zásobníku uložena adresa čítače instrukcí (PC) a program skočí na adresu vektoru přerušení 0x0004, kde je umístěna rutina pro obsluhu přerušení. Obsah používaných registrů, které mohou být během přerušení změněny je vhodné uložit na místo v paměti. Návrat z obslužné rutiny přerušení zajišťuje příkaz RETFIE. Z vrcholu zásobníku je vzata návratová hodnota a uloží se do programového čítače, program tedy může pokračovat na místě, kde skončil před vyvoláním přerušení.

8.1 Povolení přerušení

Aby mohlo přerušení nastat, musí být povoleno. Registr INTCON slouží pro povolení 4 a indikaci 3 přerušení. Všechny bity registru INTCON lze libovolně číst a nastavovat.

Tabulka 5 - Registr INTCON

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

bit 7 GIE – slouží pro globální povolení nemaskovaných (povolených) přerušení
– 1 = povolena všechna nemaskovaná přerušení, 0 = zakázána

bit 6 PEIE – povolení všech nemaskovaných přerušení od periferií
– 1 = povolena nemaskovaná přerušení od periferií, 0 = zakázána

bit 5 TMR0IE – povolení přerušení od čítače/časovače TMR0
– 1 = povoleno přerušení přetečením TMR0, 0 = zakázáno

bit 4 . RBIE – povolení přerušení změnou hodnoty jednoho z vyšších bitů portu B
– 1 = povoleno přerušení změnou hodnoty bitů portu B, 0 = zakázáno

bit 3 INTE – povolení přerušení na vývodu RB0/INT
– 1 = povoleno přerušení změnou hodnoty na RB0/INT, 0 = zakázáno

bit 2 TMR0IF – příznak přerušení od čítače/časovače TMR0
– 1 = došlo k přetečení registru TMR0, 0 = nedošlo k přetečení TMR0

bit 1 INTF – příznak vnějšího přerušení z vývodu RB0/INT
– 1 = vnější žádost o přerušení z vývodu RB0/INT, 0 = nedošlo k žádosti

bit 0 RBIF – příznak přerušení změnou hodnoty na jednom z vyšších bitů portu B
– 1 = nejméně jeden bit z bitů RB7:RB4 změnil stav
– 0 = nedošlo ke změně hodnoty ani na jednom z bitů RB7:RB4

8.2 Realizace přerušení v simulátoru HyCiSim

Metoda OnMyRequest krom jiného hlídá, zda došlo k přerušení. Pokud mikroprocesor vykonává nějaký program, je v každém instrukčním cyklu provedena instrukce uložená v paměti programu na adrese odpovídající hodnotě programového čítače (PC). Po vykonání je zkontrolován bit GIE v registru INTCON, je-li nastaven, jsou povolena přerušení globálně a vyvolá se metoda HandleInterrupts.

```
procedure OnMyRequest(ID: Integer);
var
  bytes, cycles : byte;
  i : Integer;

begin
  if PC <= (LastInstructionIndex) then
     ExecuteInstruction(PC,bytes,cycles);
```

- *Před zavoláním metody pro vykonání instrukce se nejprve testuje, jestli není programový čítač mimo rozsah adres paměti programu, na kterých jsou uloženy instrukce. Do proměnné LastInstructionIndex je v metodě LoadFile uložen index poslední instrukce nahrané v paměti programu.*

```
procedure ExecuteInstruction(index: word; bytes, cycles : byte);
    .
    .
    if CDATA[index] <> $3FFF then
      begin
        {$define execute}
        {$include instrukce.inc}
        {$undef execute}
      end;

    PC:=PC+bytes;
```

- *Pokud datové slovo v paměti programu neodpovídá hodnotě po restartu 0x3FFF (neexistující instrukce), je připojen soubor instrukcí a vykoná se instrukce odpovídající obsahu buňky na adrese programového čítače metodou popsanou v kapitole Instrukce mikrokontroléru – Dekódování instrukce. Následně je PC inkrementován.*

```
    if (GetSFR(SFR_INTCON[0]) and $80) = $80 then
      begin
        if ((GetSFR(SFR_INTCON[0]) and $10) = $10) then
          begin
```

- *Nejprve se testuje bit GIE v registru INTCON (1 = globální přerušení povoleno). Dále je testováno povolení přerušení od vnějšího obvodu, který je připojen na pinu RB0/INT (1 = povoleno). Toto přerušení může nastat náběžnou nebo sestupnou hranou a tu je vhodné testovat již v tomto místě pro případné nastavení příznaku.*

```
if ((GetSFR(SFR_OPTION_REG[0]) and $40) = $40) then
```

```
  begin
```

- *test povolení přerušení (bit INTEDG = 1 v registru OPTION) náběžnou hranou na RB0/INT*

```
if (GetSFR(SFR_PORTB[0]) and $01=1) and (Old_PORTB and $01<>1)
 then
```

- *náběžná hrana se identifikuje jako změna na pinu z hodnoty 0 na 1*

```
SetIDATA(SFR_INTCON[0], GetSFR(SFR_INTCON[0]) or $02, true);
  end;
```

- *pokud nastala náběžná hrana, nastaví se příznak přerušení bit INTF v registru INTCON a dále už může být zavolána metoda pro obsluhu přerušení HandleInterrupts*

```
    end;
  HandleInterrupts;
  end;
end;
```

8.2.1 Skok na obsluhu přerušení

V metodě HandleInterrupts jsou obsloužena jednotlivá přerušení pokud nejsou maskována (zakázána) a pokud jsou nastaveny příznaky těchto přerušení. Následující kód provádí skok na obsluhu přerušení pokud je povoleno přerušení od čítače/časovače Timer0 a je nastaven příznak přerušení od Timer0.

```
Procedure   HandleInterrupts;
begin

if ((GetSFR(SFR_INTCON[0]) and $20) = $20) and
((GetSFR(SFR_INTCON[0]) and $04) = $04) then
  begin
    StackPointer.Push(IntToStr(PC));
    PC := $0004; //adresa vektoru přerušení
  end
else //otestují se další možná přerušení
```

- *StackPointer je instance třídy, která simuluje hardwarový zásobník typu LIFO pro ukládání adres skoků a adres návratu z podprogramu (nebo přerušení). Metoda Push třídy StackPointer ukládá na vrchol zásobníku adresu aktuální pozice v paměti programu (hodnota PC). Do programového čítače je zapsána hodnota 0x0004, kde začíná vektor přerušení. V obslužné rutině je nutné použít jako první instrukci nulování příznaku přerušení, jinak program v přerušení uvízne.*

8.2.2 Návrat z obsluhy přerušení

Pro návrat z přerušení slouží instrukce RETFIE.

```
if (CDATA[index] and $3FFF) = $9 then
          begin
            {$ifdef show}
            s:=s+'RETFIE';
            {$endif}
            {$ifdef execute}
            PC := StrToInt(StackPointer.Pop) - 1;
            cycles := 2;
            {$endif}
          End
```

- *Do programového čítače je zapsána adresa z vrcholu hardwarového zásobníku StackPointer, která do něj byla uložena obslužnou metodou pro přerušení HandleInterrupts.Tato adresa odpovídá instrukci, která měla být vykonána v dalším instrukčním cyklu.*

9 Závěr

Výsledkem této práce je rozpracovaná součástka mikrokontroléru PIC16F877A. V této fázi má mikrokontrolér nadefinovanou funkční strukturu datové a programové paměti s metodami čtení a zápisu. Přístup do paměti dat je možný přímou i nepřímou adresací. Program lze do mikrokontroléru nahrát v dialogovém okně vyvolaném dvojklikem na součástku během spuštěné simulace v simulátoru HyCiSim a to v podobě hex souboru vytvořeném v návrhářském prostředí MPLAB. V okně se zobrazí překlad hex souboru na jednotlivé instrukce spolu s názvy registrů, adres či konstant.

Chod mikrokontroléru je řízen simulovaným instrukčním cyklem, který trvá čtyři takty mikroprocesoru. Během tohoto cyklu je vykonána instrukce uložená na adrese programového čítače (PC), obsluha čítačů/časovačů a obsluha přerušení.

Instrukční soubor se nachází v souboru instruction.inc ve složce Common. Kromě instrukcí SLEEP a CLRWDT obsahuje všechny instrukce (některé však nebyly dostatečně testovány a tak není zaručena naprostá bezchybnost).

Program obsahuje obvody všech tří čítačů/časovačů. Všechny tyto čítače dle datasheetu nastavují příznak přetečení a mají programovatelnou předděličku. Dle dokumentace k mikroprocesoru PIC16F877A má Timer0 volitelný zdroj hodinového signálu (vnitřní/vnější) a volitelnou aktivní hranu vnějšího hodinového signálu. Timer1 má také volitelný zdroj hodinového signálu a může fungovat v synchronním i asynchronním režimu. Timer2 je vybaven dvěma předděličkami a nastavitelnou periodou pomocí registru PR2.

Obsluhu přerušení mohou vyvolat obvody již zmíněných čítačů/časovačů nebo změna na pinu RB0/INT vnějším obvodem. Přerušení lze zakázat globálně nebo maskovat jednotlivá přerušení. Pro skok na vektor přerušení a jiné skokové instrukce je naimplementován osmi-úrovňový zásobník typu LIFO (třída LIFO/stack.pas).

Projekt simulace mikrokontroléru PIC16F877A tedy nabízí pokračování v rámci diplomové práce. Ta by se zaměřila na dokončení zatím nerealizovaných integrovaných periferií (PWM, A/D převodník, USART a další), důkladnější testování instrukcí a testování simulace v HyCiSim simulátoru vůči reálnému mikrokontroléru. Výsledkem toho by studenti TUL dostali do ruky nástroj, který je zcela zdarma a umožňuje simulaci vývojových přípravků používaných zejména v předmětu PMP a PHS.

10 Seznam použité literatury

1 Microchip Technology Incorporated [online]. PIC16F87XA Data Sheet. Arizona, Chandler, USA. 07/28/2003. DS39582B. Dostupné na World Wide Web: <http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf>.

2 Microchip Technology Inc. 2003. Jednočipový mikropočítač PIC16F87xA. Ing. Vladimír Čebiš, Ing. Petr Vaňous, 2005 [online]. Dostupné na World Wide Web: <uloziste.micovo.cz/Datasheety/PIC16F87xA.pdf>.

3 Kicad/file formats [online], poslední aktualizace 18 March 2012, at 06:33 [cit. 16. 5. 2012]. Wikipedie. Dostupné z World Wide Web: <http://en.wikibooks.org/wiki/Kicad/file_formats>

11 Přílohy

K bakalářské práci je přiložen CD-ROM obsahující:

- digitální kopii bakalářské práce ve formátu PDF
- simulátor HyCiSim ve složce HyCiSim, kód součástky mikrokontroléru PIC16F877A je v souboru: HyCiSim/Common/PartPIC16F877A.pas, instrukční soubor je uložen v souboru HyCiSim/Common/instrukce.inc